Keystroke Dynamics (Algorithm Analysis)

CS20B1014 (SANJAY) **CS20B1019 (**JAYANTH) **CS20B1029 (**SHYAM SUNDAR) **CS20B1031 (**DAATHWINAAGH)

Abstract*—This research study investigates the effectiveness and performance of keystroke dynamics algorithms in user authentication systems. Keystroke dynamics refers to the unique typing patterns and rhythms exhibited by individuals during the input of text. The objective of this study was to evaluate the accuracy and performance of various keystroke dynamics algorithms and determine their suitability for practical implementation.

The CMU Keystroke Dynamics Dataset, also known as the CMU-Multi-Modal Behavioral Biometrics (MMBB) Keystroke Dataset, is a widely used dataset in the field of keystroke dynamics research. It was developed and made publicly available by the Carnegie Mellon University (CMU) in Pittsburgh, Pennsylvania. The data consist of keystroke-timing information from 51 subjects (typists), each typing a password (`.tie5Roanl`) 400 times.

Link to dataset : [https://www.cs.cmu.edu/~keystroke/]

The accuracy of each algorithm was assessed by measuring two key metrics: ***false acceptance rate (FAR)*** and ***false rejection rate (FRR)*** and the performance of algorithm is measured by ***equal error rate*** and ***accuracy*** of the algorithm. Additionally, the computational performance of the algorithms was evaluated in terms of processing time and resource utilization. The experimental results were statistically analyzed to identify significant variations among the different algorithms.

The findings of this research, based on the CMU Keystroke Dynamics Dataset, indicate that keystroke dynamics algorithms can achieve a high level of accuracy in distinguishing between genuine users and imposters. The best-performing algorithms achieved *high accuracy and low equal error rate (EER)*, thereby demonstrating their ability to effectively authenticate users based on their typing patterns. Furthermore, the computational performance analysis revealed that the selected algorithms demonstrated efficient resource utilization and processing times suitable for real-time applications.

The utilization of the CMU Keystroke Dynamics Dataset in this research enhances its credibility and generalizability. The dataset, consisting of a diverse range of participants, provides a representative sample for evaluating the algorithms' performance. The outcomes of this research have significant implications for the development of secure and user-friendly authentication systems, leveraging the CMU dataset as a benchmark.

Overall, this research contributes to the advancement of keystroke dynamics technology using the CMU Keystroke Dynamics Dataset and its potential for enhancing user authentication. Further research can focus on exploring additional factors that may impact the performance of these algorithms, such as user fatigue, typing variations over time, or the incorporation of other biometric modalities, to further refine and improve the accuracy and reliability of keystroke dynamics-based authentication systems.

I. NEED OF KEYSTROKE BASED AUTHENTICATION

Keystroke authentication systems provide an additional layer of security and offer several benefits in various contexts. Here are some key reasons highlighting the need for keystroke authentication systems:

Enhanced Security: Keystroke dynamics analysis adds an extra level of security to traditional authentication methods such as passwords or PINs. It leverages the unique typing patterns and rhythms of individuals, making it difficult for unauthorized users to mimic or replicate accurately. This biometric factor increases the overall security of the authentication process, reducing the risk of unauthorized access to sensitive information or systems.

Non-Intrusive and user-friendly: Keystroke authentication systems are non-intrusive and do not require additional hardware or sensors. Users can be authenticated based on their typing patterns without any significant changes to their typical interaction with devices. This user-friendly approach eliminates the need for complex and time-consuming authentication procedures, improving user experience and reducing resistance towards security measures.

Continuous authentication: Keystroke dynamics can be continuously monitored during a user's session, providing a form of continuous authentication. This feature ensures that the user's identity remains verified throughout their interaction with a system or application, offering increased security against unauthorized access in cases where a user may inadvertently leave their device unattended or an unauthorized individual gains physical access to the device.

Biometric soofing resistance: Keystroke dynamics analysis is inherently difficult to spoof compared to other biometric modalities. For example, fingerprint or face recognition systems can be tricked using replicas or photographs, but replicating someone's typing patterns accurately is considerably more challenging. This resistance to spoofing makes keystroke authentication systems more reliable and secure against fraudulent attempts.

Cost-Effective Implementation: Keystroke authentication systems can be implemented without significant additional costs, as they utilize existing keyboard infrastructure and do not require specialized hardware or sensors. This cost-effectiveness makes them a viable security solution for organizations that may not have extensive resources to invest in more complex authentication methods.

Behavioral Insights: Keystroke dynamics analysis can provide valuable insights into user behavior and typing patterns. This data can be utilized for various purposes, such as identifying anomalies or detecting changes in user behavior, which may indicate potential security threats or health-related concerns. The behavioral insights derived from keystroke analysis can contribute to proactive security measures and user monitoring.

II. Development of keystroke authentication

The process of keystroke dynamics involves several steps to capture, analyze, and utilize the typing patterns and rhythms of individuals for authentication or other purposes. Here is an overview of the typical process of keystroke dynamics:

***Data Collection*:** The first step is to collect keystroke data from individuals. This can be done by designing a data collection task where participants are asked to type specific texts or perform certain typing exercises. The data collection task should be designed to capture a sufficient number of keystrokes to establish a representative sample of each user's typing behavior.

Keystroke Feature Extraction: Once the keystroke data is collected, relevant features need to be extracted from the captured keystrokes. These features typically include timing-based measurements, such as key press durations, key flight times (time between consecutive key presses), and key digraphs (time between the release of one key and the press of the next key). Other features, such as the dwell time (time between key press and release) and the latency between specific key pairs, may also be extracted depending on the specific application.

Feature Preprocessing: The extracted features may undergo preprocessing steps to remove noise, normalize the data, or reduce the dimensionality. Preprocessing techniques can include outlier removal, data normalization, feature scaling, or feature selection methods to enhance the quality and efficiency of subsequent analysis steps.

Algorithm Development: Keystroke dynamics algorithms are developed to analyze and model the extracted features. Various machine learning and statistical techniques can be employed, such as decision trees, support vector machines (SVM), neural networks, or statistical models like Hidden Markov Models (HMM) or Gaussian Mixture Models (GMM). These algorithms are trained on a labeled dataset, consisting of known genuine and imposter keystroke samples, to learn the patterns and characteristics of each user's typing behavior.

Model Training and Evaluation: The developed keystroke dynamics algorithm is trained using the labeled dataset. The training process involves optimizing the algorithm's parameters to minimize false acceptance and false rejection rates, balancing between security and usability. The algorithm's performance is evaluated using evaluation metrics such as the false acceptance rate (FAR), false rejection rate (FRR), receiver operating characteristic (ROC) curves, or equal error rate (EER).

Deployment and Authentication: Once the keystroke dynamics algorithm is trained and evaluated, it can be deployed for authentication purposes. During authentication, a user's keystrokes are captured in real-time or compared against the stored model. The algorithm analyzes the captured keystrokes and computes a similarity score or probability indicating the likelihood of the user being genuine. Based on a predetermined threshold, the user is either accepted or rejected.

Continuous Monitoring and Adaptation: In some cases, keystroke dynamics systems incorporate continuous monitoring to detect changes in typing behavior over time. This allows for adaptive authentication or alerting systems, where significant deviations from the established typing patterns can trigger additional security measures or re-authentication requests.

III. Experimenting on the data

A. *Data collection*

The CMU Keystroke Dynamics Dataset, also known as the CMU-Multi-Modal Behavioral Biometrics (MMBB) Keystroke Dataset, is a widely used dataset in the field of keystroke dynamics research. It was developed and made publicly available by the Carnegie Mellon University (CMU) in Pittsburgh, Pennsylvania. The data consist of keystroke-timing information from 51 subjects (typists), each typing a password (`.tie5Roanl`) 400 times.

Link to dataset : [https://www.cs.cmu.edu/~keystroke/]

B. *Keystroke features and their extraction*

Keystroke dynamics analysis involves extracting various features from keystroke data to characterize an individual's typing behavior. These features capture the timing and rhythm of key press events and can be used to distinguish between different users. Here are some common features used in keystroke dynamics:

- *Key Press Duration*: It refers to the time duration between the key press and key release events. This feature measures the duration for which a key remains pressed before being released.
- *Key Flight Time*: It is the time duration between the release of one key and the press of the next key. Key flight time captures the temporal gap between consecutive keystrokes.
- *Key Dwell Time*: It represents the duration between the press and release events of a single key. Dwell time provides insights into the duration for which a user holds down a particular key.
- *Key Digraph Time:* It measures the time duration between the release of one key and the subsequent press of another key. Key digraph time captures the timing relationship between consecutive key pairs.
- *Inter-Key Time:* This feature represents the time duration between the press of one key and the subsequent press of another key, regardless of the release events. It measures the temporal gap between any two consecutive key presses.
- *Latency:* Latency refers to the time delay between the initiation of a stimulus (e.g., appearance of a prompt) and the subsequent key press event. It measures the time taken by a user to respond to a specific stimulus.
- *Hold Time:* Hold time represents the duration for which a user holds down a specific key without releasing it. It characterizes the continuous depression of a key.
- *Release Time:* Release time captures the time duration between the release of a key and the subsequent press of another key. It measures the temporal gap between the release and subsequent press events.
- *Typing Speed:* Typing speed indicates the rate at which a user types. It is often measured in terms of words per minute (WPM) or characters per minute (CPM).

IV. Algorithms and measuring the accuracy

The following are the five machine learning algorithms are developed tested, measured accuracy and equal error rate.

1. K Means Clustering

2. Support Vector Machine (SVM)
3. K Nearest Neighbours classifier
4. Manhattan distance classifier
5. Neural Networks

K Means Clustering Algorithm

K-means clustering is an unsupervised machine learning algorithm used for grouping similar data points into clusters. It aims to partition a given dataset into K clusters, where K is a user-defined parameter representing the number of clusters to be created. The algorithm is iterative and seeks to minimize the sum of squared distances between the data points and their respective cluster centroids.

Initialization: Randomly select K data points from the dataset as initial cluster centroids.

Assignment: Assign each data point to the nearest centroid based on the Euclidean distance or any other distance metric.

***Update*:** Recalculate the centroids of each cluster by computing the mean of all the data points assigned to that cluster.

Repeat Steps 2 and 3: Iterate the assignment and update steps until convergence or a predefined number of iterations is reached. Convergence occurs when the centroids no longer change significantly or the assignments stabilize.

Termination: Once convergence is reached, the algorithm terminates, and the final cluster centroids and assignments are obtained.

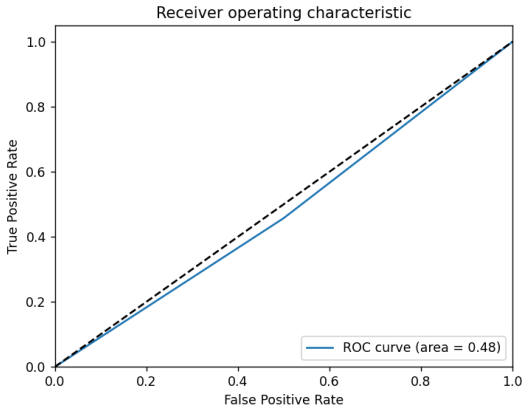
Firstly. load the CMU keystroke dynamics dataset and handles missing data using SimpleImputer from sklearn.impute then splits the data into training and testing sets and separates the features (keystroke timings) from the labels (user identity). Next, convert the labels to binary values based on a threshold. Then k-means clustering is applied to the training features with two clusters, uses the trained k-means model to predict the labels of the testing data, and converts predicted labels to binary values.

calculate performance metrics such as confusion matrix, accuracy, precision, recall, F1-score, and root mean squared error, as well as the ROC curve and AUC. Finally the performance metrics and plots the ROC curve are printed using matplotlib.

Performance of K means Algorithm (measured)

Accuracy	0.48049019607843135 or **48%**
Precision	0.43566176470588236
Recall	0.4568430070678946
F1-score	0.4460010454783063
RMS error	0.7207702851266613
ROC AUC	0.47864750244915244

ROC Curve (K Means clustering)



Support vector machine

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is particularly effective when working with high-dimensional data. SVMs can be used for both linear and non-linear classification and regression problems.

The main idea behind SVM is to find an optimal hyperplane that separates data points of different classes with the largest possible margin. The hyperplane is determined by a subset of training samples called support vectors. These support vectors are the data points closest to the decision boundary.

SVM aims to find the optimal hyperplane by solving an optimization problem. The objective is to maximize the margin (distance) between the hyperplane and the support vectors while minimizing the classification error. This is done by solving a convex quadratic programming problem.

***Decision boundary and classification*:** Once the optimal hyperplane is found, new data points can be classified based on their position relative to the decision boundary. Data points on one side of the hyperplane are assigned to one class, while those on the other side are assigned to the other class.

Accuracy	0.022
Equal Error Rate	0.340
Sensitivity	1.000
Root Mean Square Error	18.210

K Nearest Neighbours Algorithm

The k-Nearest Neighbors (k-NN) algorithm is a supervised machine learning algorithm used for both classification and regression tasks.

Data preparation: Begin with a labeled dataset consisting of feature vectors and their corresponding class labels.

Distance calculation: Compute the distance between the input data point and all the training data points. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance.

Finding nearest neighbors: Select the k nearest neighbors to the input data point based on the calculated distances. The value of k is a user-defined parameter.

Majority voting: For classification, determine the class label of the input data point based on the majority class among its k nearest neighbors. Each neighbor's contribution can be weighted based on its distance or given equal weight.

Prediction: Assign the predicted class label to the input data point based on the majority voting result.

Training and Testing data is split. The k-NN classifier is trained on the training set using the KNeighborsClassifier class from scikit-learn with k=3, i.e., it considers the 3 nearest neighbors. The test data is run and performance metrics such as Accuracy, confusion matrix, Mean square error are calculated. *Mean Squared Error: 368.97867647058825*

	Precision	*recall*	*F1 score*	*support*
Accuracy			0.02	4080
Macro avg	0.02	0.02	0.01	4080
Weighted avg	0.02	0.02	0.01	4080

Manhattan distance classifier

The Manhattan distance, also known as the L1 distance or city block distance, measures the absolute differences between coordinates of two points in a vector space. It is calculated as the sum of the absolute differences between the corresponding features of the two points.

To utilize the Manhattan distance in the k-NN algorithm for classification, you would follow the general steps of the k-NN algorithm but use the Manhattan distance metric instead of other distance metrics like Euclidean distance.

Distance calculation: Compute the Manhattan distance between the input data point and all the training data points. Manhattan distance between two points (x1, y1) and (x2, y2) in a 2D space is given by |x2 - x1| + |y2 - y1|. For higher-dimensional spaces, you would calculate the sum of the absolute differences between the corresponding coordinates.

Finding nearest neighbors: Select the k nearest neighbors to the input data point based on the calculated Manhattan distances.

Majority voting: Determine the class label of the input data point based on the majority class among its k nearest neighbors.

Prediction: Assign the predicted class label to the input data point based on the majority voting result.

Performance metrics :

Equal Error Rate	*0.12*
Accuracy	***67%***

Neural networks

Neural networks, also known as artificial neural networks or simply neural nets, are a class of machine learning algorithms inspired by the structure and functioning of the human brain. They are widely used for various tasks such as pattern recognition, classification, regression, and more.

At a high level, a neural network consists of interconnected nodes called neurons, organized in layers. These layers include an input layer, one or more hidden layers, and an output layer. Each neuron receives inputs, performs a computation, and produces an output that is passed on to the next layer.

Neuron (or Perceptron): The basic computational unit of a neural network. It receives inputs, applies a weighted sum and an activation function, and produces an output.

Activation Function: Non-linear functions applied to the output of a neuron. Activation functions introduce non-linearity into the network, allowing it to learn complex patterns. Examples include the sigmoid, ReLU, and tanh functions.

Weights and Biases: Each neuron has associated weights that are multiplied with the inputs and summed to produce an output. Biases are additional parameters added to the weighted sum. These weights and biases are adjusted during the training process to optimize the network's performance.

Forward Propagation: The process of feeding input data through the network, layer by layer, to produce an output prediction. In each layer, the outputs of the previous layer are used as inputs to the next layer.

Loss Function: A function that measures the discrepancy between the predicted output of the neural network and the expected output (ground truth). The choice of loss function depends on the task, such as mean squared error for regression or cross-entropy for classification.

Backpropagation: An algorithm used to train the neural network by adjusting the weights and biases based on the computed loss. It calculates the gradients of the loss function with respect to the weights and biases, allowing the network to update them using optimization techniques like gradient descent.

Training and Optimization: The process of iteratively adjusting the network's weights and biases to minimize the loss function. The goal is to find the optimal set of weights and biases that make the network produce accurate predictions.

The data is split into training and testing sets using the sample function, with 80% of the data used for training and the remaining 20% for testing. The features (input data) are then scaled using StandardScaler. The scaled training and testing features and corresponding labels are stored in separate variables.

The neural network is defined using Keras Sequential API. It consists of three layers: two hidden layers with 64 neurons each and ReLU activation, and one output layer with a single neuron and sigmoid activation. The model is then compiled with the binary cross-entropy loss function, Adam optimizer, and accuracy as the evaluation metric.

The model is trained on the training data for 10 epochs with a batch size of 32. After training, the model is evaluated on the testing set, and the accuracy is printed.

Finally, the true positive rate (sensitivity), false positive rate (specificity), and equal error rate (EER) are computed using the predictions made by the model on the testing set. The results are obtained.

Performance metrics :

Equal Error Rate	0.78
Accuracy	23%

CONCLUSION AND REFERENCES

Thus by comparing the algorithms. ***Manhattan distance*** classifier is best suitable for the case with keystroke based authentication.

[1] *Kevin S. Killourhy and Roy A. Maxion. "Comparing Anomaly Detectors for Keystroke Dynamics," in Proceedings of the 39th Annual International Conference on Dependable Systems and Networks (DSN-2009), pages 125-134, Estoril, Lisbon, Portugal, June 29-July 2, 2009. IEEE Computer Society Press, Los Alamitos, California, 2009. (pdf)*

[2] *T. Sing, O. Sander, N. Beerenwinkel, T. Lengauer. "ROCR: visualizing classifier performance in R," Bioinformatics 21(20):3940-3941 (2005). (link)*

[3] https://www.cs.cmu.edu/~keystroke/